SANDIA REPORT

SAND2015-0577
Unlimited Release
Printed Feburary 2015

Portable File Format (PFF) specifications

Daniel H. Dolan

Sandia National Laboratories

SAND2015-0577
Unlimited Release
Printed Feburary 2015

Portable File Format (PFF) specifications

Daniel H. Dolan
Sandia National Laboratories
P.O. Box 5800, M.S. 1189
Albuquerque, NM 87185

Abstract

Created at Sandia National Laboratories, the Portable File Format (PFF) allows binary data transfer across computer platforms. Although this capability is supported by many other formats, PFF files are still in use at Sandia, particularly in pulsed power research. This report provides detailed PFF specifications for accessing data without relying on legacy code.

Acknowledgments

The Portable File Format was created at Sandia National Laboratories by Dave Seidel. Most of the PFF development team has retired from Sandia, and some format details were thought to be lost. Information provided by Tim Pointon prevented this loss from happening entirely. Source code from the Hermes Utilities project and advice from Kyle Cochrane were also critical. Ray Lemke and Tommy Ao provided files for developing this report and served as beta testers of the new PFF writer. Jean-Paul Davis helped track down bugs in this writer.

Contents

Figures

1 Introduction

The Portable File Format (PFF) was created at Sandia National Laboratories as part of the QUICKSILVER family of codes.[1] The binary file format was developed to share data on different computer systems, which at the time (mid-1980s) was no small feat. Modern file formats (such as HDF[2]) address this need in a more general manner, but there is a significant PFF legacy at Sandia and in pulsed power research. The PFIDL package,[3] a Sandia-developed expansion to IDL (Interactive Data Language), uses PFF files extensively.

Although Fortran and C routines for PFF management are available as part of the Hermes Utilities project (`hermes-util.sourceforge.net`), file format specifications are sparse. An unpublished report[4] provides the general structure of PFF, but file access is described mostly in terms of legacy Fortran routines. No formal PFF documentation seems to exist.

The need for reading PFF files into MATLAB became apparent during the development of the SMASH (Sandia Matlab AnalysiS Hierarchy) toolbox.[5] To address this need, a detailed format specification was developed from legacy routines, limited documentation, and considerable experimentation. Results from that effort are summarized in this report.

2 Format overview

PFF files store groups of related information, or datasets, in a binary format (big-endian convention). Files typically use the `*.pff` extension, though this may not be strictly enforced. Figure 1 shows the conceptual layout of a PFF file containing two datasets. Headers at the beginning of the file and each dataset describe the file's organization. Every entry in the file (including the headers) is made using the fundamental types shown in Figure 2. Appendix A provides technical details for each entry type.

Files begin with a 16 word (32 byte) file header. The first word (FFRAME) must be equal to -4 for a valid PFF file. The second word indicates the closure state: if equal to -3, the file was not formally closed and directory datasets are not available; otherwise, combining words 2–4 into a long word reveals the location of the first directory data set (in words). All remaining words of the file header are typically -3 and can be ignored.

Datasets are located immediately after the file header. Each dataset begins with a start word (DFRAME) equal to -1. Immediately after the start word is a dataset header with this layout.

- Length of dataset LDS (long word) in words
- Dataset type TRAW (word)
- Dataset type version VDS (word)
- Application type TAPP (word)
- Reserved for future use RFU (10 words)
- Dataset label TYPE (string)
- Dataset title TITLE (string)

Dataset content begins immediately after this header. Advancing the file pointer from the position prior to reading DFRAME by $2\times$LDS bytes moves to the end of dataset. The next word read from the file will either be -1 (start word for next dataset) or -2 (stop word, no more datasets).

Formally closed PFF files contain directory datasets after the stop word. Directory datasets describe the type, length, and location of datasets prior to the stop word. This information allows non-sequential dataset access.

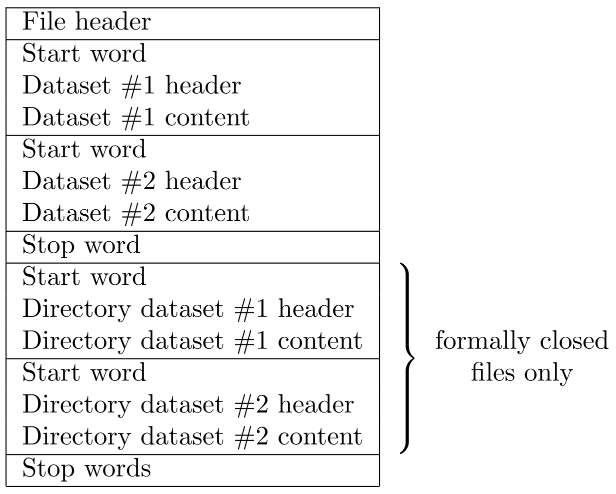

Figure 1. Conceptual file layout with two datasets

Word One 16-bit signed integer

Long word Sequence of 16-bit signed integers that represent a long integer

String Sequence of ASCII characters

Float Sequence of 16-bit signed integers that represent a floating point number

Integer array Array of 16-bit signed integers

Float array Array of floating point numbers

Figure 2. Fundamental entry types. Details of each type are given in Appendix A.

Non-sequential access

- Read file header words 2–4 to determine the first directory dataset location.
- Starting from the first directory dataset, scan until the N-th directory dataset is reached.
- Use the N-th dataset directory to locate the N-th dataset.

PFF files that are not formally closed lack directory datasets, so sequential access is required.

Sequential access

- Read past the file header to the start word.
- Read the current dataset header. If this is N-th dataset, read the content and close the file. Otherwise, jump over the content.
- Continue until the N-th dataset or the stop word is found.

Legacy programs support both access methods.

A series of stop words after the last dataset terminates a PFF file. A single stop word appears to be sufficient for 32-bit legacy compilations, but additional stop words are needed on 64-bit legacy compilations.[1] Terminating the file with at least 1024 stop words (after the actual stop word) keeps the file pointer from moving beyond the end of file marker; in legacy codes, that operation triggers a repair process where datasets may be lost.

3 Reading datasets

Dataset access varies with type, which is specified in the TRAW entry of their header. Ten dataset types are defined in PFF.

Directory datasets are defined by TRAW=0.

Uniform 3D floating point datasets are defined by TRAW=1.

Uniform 1D floating point datasets are defined by TRAW=2.

Non-uniform 3D floating point datasets are defined by TRAW=3.

Non-uniform 3D vector floating point datasets are defined by TRAW=4.

Multi-dimensional vertex datasets are defined by TRAW=5.

Integer/float list datasets are defined by TRAW=6.

Multi-dimensional vectors on multi-dimensional space datasets are defined by TRAW=7.

Non-uniform 3D grid datasets are defined by TRAW=8.

Non-uniform 3D integer datasets are defined by TRW=9.

Directory datasets (TRAW=0, also known as PFTDIR) are organized as follows.

- Dataset type TRAW (word)

[1]PFIDL on OS X and Windows currently uses a 32-bit compilation, while the current Linux is 64-bit.

- Directory length LENDIR (long word)
- Directory location LOCDIR (long word)

Uniform 3D floating point datasets (TRAW=1, also known as PFTUF3) are organized as follows.

- Number of blocks NBLOCKS (word)
- Blocks are stored sequentially in the following formation.
  - Number of grid points NX (long word)
  - Number of grid points NY (long word)
  - Number of grid points NZ (long word)
  - Application-specific values ISPARE (5 words)
  - Initial point X0 (float)
  - Spacing DX (float)
  - Initial point Y0 (float)
  - Spacing DY (float)
  - Initial point Y0 (float)
  - Spacing DZ (float)
  - Axes label XLABEL (string)
  - Axes label YLABEL (string)
  - Axes label ZLABEL (string)
  - Block label BLABEL (string)
  - Data array FARRAY (float array)

Uniform 1D floating point datasets (TRAW=2, also known as PFTUF1) are organized as follows.

- Number of blocks NBLOCKS (word)
- Blocks are stored sequentially in the following formation.
  - Number of points NX (long word)
  - Application-specific values ISPARE (5 words)
  - Initial grid point X0 (float)
  - Grid spacing DX (float)
  - Grid label XLABEL (string)
  - Block label BLABEL (string)
  - Data FARRAY (float array)

Non-uniform 3D floating point datasets (TRAW=3, also known as PFTNF3) are organized as follows.

- Number of blocks NBLOCKS (word)
- Blocks are stored sequentially in the following formation.
  - Number of grid points NX (long word)
  - Number of grid points NY (long word)

  - Number of grid points NZ (long word)
  - Application-specific values ISPARE (5 words)
  - Grid array X (float array)
  - Grid array Y (float array)
  - Grid array Z (float array)
  - Grid label XLABEL (string)
  - Grid label YLABEL (string)
  - Grid label ZLABEL (string)
  - Block label BLABEL (string)
  - Data array FARRAY (float array)

Non-uniform 3D vector floating point datasets (TRAW=4, also known as PFTNV3) are organized as follows.

- Number of blocks NBLOCKS (word)
- Blocks are stored sequentially in the following formation.
  - Number of grid points NX (long word)
  - Number of grid points NY (long word)
  - Number of grid points NZ (long word)
  - Application-specific values ISPARE (5 words)
  - Grid array X (float array)
  - Grid array Y (float array)
  - Grid array Z (float array)
  - Grid label XLABEL (string)
  - Grid label YLABEL (string)
  - Grid label ZLABEL (string)
  - Block label BLABEL (string)
  - Vector component VX (float array)
  - Vector component VY (float array)
  - Vector component VZ (float array)

Multi-dimensional vertex data with attributes (TRAW=5, also known as PFTVTX) are organized as follows.

- Vertex dimensionality M (word)
- Attribute dimensionality N (word)
- Number of vertices NV (long word)
- Application-specific values ISPARE (5 words)
- Vertex coordinate labels VLABEL (M strings)
- Attribute labels ALABEL (N strings)
- Version-specific vertex list
  - If VDS is -3 and $M>0$, a 2D vertex list ($M\times NV$) is read (float array)

  - If VDS is 1, a 1D vertex list (1×NV) is read (float array)

- Attribute list AI (float array)

Integer/float list dataset (PFTIFL, also known as TRAW=6)

- Float flag FLTFLG (word)
- Float list length NFL (long word)
- Integer array IARRAY (integer array)
- Float list FLIST (NFL floats)
- If FLTFLG is not zero, another array FARRAY is read (float array)

Multi-dimensional vectors on a multi-dimensional space datasets (TRAW=7, also known as PFTNGD) are organized as follows.

- Space dimensionality M (word)
- Vector dimensionality N (word)
- Version-specific grid points NX
  - If VDS=1, NX is read as M long words
  - Otherwise NX is read as M words
- Application-specific values ISPARE (integer array)
- Axes labels ALABEL (M strings)
- Vector component labels VLABEL (N strings)
- $i = 1..M$ grid point arrays X_i (M floating arrays)
- $j = 1..N$ vector component arrays Y_j (N floating arrays)

Non-uniform 3D grid datasets (TRAW=8, also known as PFTNG3) are organized as follows.

- Number of blocks NBLOCKS (word)
- Blocks are stored sequentially in the following formation.
  - Number of grid points NX (long word)
  - Number of grid points NY (long word)
  - Number of grid points NZ (long word)
  - Application-specific values ISPARE (integer array)
  - Grid array X (float array)
  - Grid array Y (float array)
  - Grid array Z (float array)
  - Grid label XLABEL (string)
  - Grid label YLABEL (string)
  - Grid label ZLABEL (string)

  - Block label BLABEL (string)

Non-uniform 3D integer data (TRAW=9, also known as PFTNI3) are organized as follows.

- Number of blocks NBLOCKS (word)
- Blocks are stored sequentially in the following formation.
  - Number of grid points NX (long word)
  - Number of grid points NY (long word)
  - Number of grid points NZ (long word)
  - Application-specific values ISPARE (integer array)
  - Grid array X (float array)
  - Grid array Y (float array)
  - Grid array Z (float array)
  - Grid label XLABEL (string)
  - Grid label YLABEL (string)
  - Grid label ZLABEL (string)
  - Block label BLABEL (string)
  - Data array IARRAY (integer array)

Five of the original six PFF datasets (TRAW=1–6) reserve five words for application-specific values (ISPARE). The PFTNGD dataset (TRAW=7) changed the ISPARE entry to an integer array of arbitrary length, and this convention was carried over to subsequent datasets (TRAW=8–9). ISPARE values are not required for read access, but the developers should be aware of potential size variation between dataset.

4 File access

Reading from a PFF file is straightforward with sequential access.

- Verify format
  The first word of the file header must be -4. Other values indicate a non-PFF file or invalid endian convention.
- Locate dataset
  After reading the file header, the n-th dataset is reached by reading $n-1$ dataset headers and skipping these datasets. Datasets can be selected by the TYPE/TITLE entries, but there is no requirement for these vales to be unique.
- Extract dataset
  The n-th dataset is read from the file based on the format specified in the header (TRAW).

Creating new files or appending to existing files is considerably more complicated. As a general rule, these operations are not recommended outside of legacy PFF codes. A limited exception to this recommendation is given below.

The SMASH toolbox provides a `readFile` function (part of the `FileAccess` package) that accepts PFF files.

```
>> record=SMASH.FileAccess.readFile(filename,[format],[dataset]);
```

The file name (*.pff extension required) is only required input: the format defaults to 'pff' if omitted or empty, and the first dataset is automatically returned if no specific request is made. Any dataset in the file can be requested by number.

```
>> record=SMASH.FileAccess.readFile(filename,'pff',2); % load second dataset
```

The `probeFile` function reveals the contents of a PFF file to assist dataset selection.

```
>> probeFile(filename); % numbered dataset list in command window
```

For more information on these capabilities, refer to the online documentation by typing "`doc SMASH`" in the command window.

More advanced PFF capabilities are provided by the `PFFfile` class, which is also part of the `FileAccess` package.

```
>> object=PFFfile(filename);       % create object
>> object=select(object,filename); % change PFF file associated with object
>> probe(object);                  % list datasets in command window
>> probe(object,'gui');            % list datasets in separate window
```

The class also provides limited write support.

```
>> write(object,data);
```

The input "data" is a MATLAB structure used to construct a PFTNGD dataset. Structure fields "Grid" and "Vector" must be specified to define space and vector dimensionality, respectively; optional grid, vector, type, and text labels are also permitted. Several output modes are supported.

```
>> write(object,data,'create');    % create new file, throw error if file already exists
>> write(object,data,'overwrite'); % create new file, erase existing file as needed
>> write(object,data,'append');    % append dataset to existing file
```

Directory datasets are not currently generated by SMASH. Instead, PFF files are terminated by a continuous block of stop words after the last dataset.

The `PFFfile` class is intended for advanced users and SMASH developers. Standard PFF read/probe operations should be performed with `readFile`/`probeFile` functions described above. PFF write operations should be managed with the ("export") method provided by the various classes in SMASH (`Signal`, `Image`, etc.).

5 Summary and recommendations

Detailed specifications for the Portable File Format (PFF) are now available. Using this information, PFF files can be read without relying on legacy code. This documentation was developed with MATLAB, but the results can be applied in any computer language.

The information provided here is sufficient for probing and reading existing files; limited file creation is also possible. The SMASH toolbox can read virtually any PFF file, but its write capabilities are restricted to PFTNGD datasets with sequential access (directory datasets are not written or preserved). These capabilities were designed for sharing data between PFIDL and MATLAB, but should work with any application where PFF is used.

The capabilities provided by PFF have been superseded by several standard formats, notably HDF5. As such, PFF files are not recommended for use outside of legacy applications. A transition from PFF to modern storage formats is strongly suggested.

References

[1] D.B. Seidel, R.S. Coats, M.L. Kiefer, T.D. Pointon, and L.P. Mix. PFF–a compact, machine-independent file format for simulation data. In *9th Biennial CUBE Symposium*, (1990). SAND90-1399a.

[2] The HDF Group. Hierarchical Data Format, version 5, (1997–2014). `http://www.hdfgroup.org/HDF5/`.

[3] L.P. Mix, R.S. Coats, and D.B. Seidel. PFIDL: procedures for the analysis and visualization of data arrays. In *Tri-Laboratory Engineering Conference on Computational Modeling*, (1995).

[4] D.B. Seidel. PFF–a Portable File Format users's guide. Technical report, (2008, unpublished).

[5] D.H. Dolan and T. Ao. The Sandia Matlab AnalysiS Hierarchy (SMASH) package. Technical Report (in preparation), Sandia National Laboratories.

A PFF data types

This appendix describes the fundamental data types found in PFF files. Emphasis is placed on how data is read from a file: organization, conversion, and so forth. Limited discussion for writing certain data types to a PFF file is also provided. Note that all read/write operations must be performed in big-endian convention.

Words are signed 16-bit integers. Standard binary read and write operations may be used.

Long words are integers represented by a sequence of 16-bit signed integers. The conversion of three 16-bit integers (words) I_1, I_2, and I_3 to a long integer I is:

$$M_1 \equiv 2^{14} \quad M_2 \equiv 2^{15} \quad M_3 \equiv 2^{15}$$
$$s = \begin{cases} +1 & I_1 < M_1 \\ -1 & \text{otherwise} \end{cases}$$
$$I = s\left(\left(\text{mod}(I_1, M_1)M_2 + I_2\right)M_3 + I_3\right).$$

The reverse conversion requires a loop calculation. Starting with $R = |I|$, the values of I_k are determined in reverse order ($k = 3, 2, 1$).

If $R \leq M_k$:

$$Q = \text{floor}\left(\frac{R}{M_k}\right), \qquad I_k = R - QM_k, \qquad R = Q$$

Else:

$$I_k = Q, \qquad R = 0$$

The final value of R must be zero for a valid conversion; otherwise, I is too large to be represented with the three words ($|I| > 2^{44} - 1$). The first word is modified ($I_1 = I_1 + M_1$) when $I < 0$.

Strings are sequences of ASCII characters. Strings begin with a word that specifies string length (in words), followed by a set of 8-bit characters. The string may contain an extra character, usually a space, to ensure that the string spans an even number of characters for storage as an integer number of words.

Floats are a floating point numbers represented by a sequence of 16-bit signed integers. The conversion of three 16-bit integers (words) I_1, I_2, and I_3 to a float F is:

$$F = (1 - 2\ \mathrm{mod}(I_3, 2)) \big((I_2 \times 2^{-15} + I_1)2^{-15} + 1\big) \times 2^{(I_3/2)-8193}.$$

The reverse conversion is only required for writing uniform (PFTUF1 and PFTUF3) and the float list (PFTIFL) datasets. These datasets should not be created outside of legacy software.

Integer arrays are sequences of 16-bit signed integers. Arrays begin with one long word that specifies the array length, followed by a set of 16-bit signed integers. Standard binary read and write operations may be used.

Float arrays are sequences of floating point number. Float arrays can be defined with different precisions determined by the first word:

- If the first word equals -6, an unused word and a long word (array length M) are read from the file. Immediately afterwards are M 32-bit (single-precision) floating point numbers.
- If the first word is not equal to -6, the file pointer is moved back one word. Two floats (offset F_0 and scale s) are read from the file, followed by a long word for the array length. This many words are then read as 16-bit signed integers I_k and converted to floating point numbers ($F_k = F_0 + sI_k$).

Outside of legacy code, float arrays should always be written with the 32-bit precision.

DISTRIBUTION:

1 L.P. Mix
lpaulmix@gmail.com

1 D.B. Seidel
dbseidel@comcast.net

1	MS 1106	T. Ao, 1646
1	MS 1106	A. Harvey-Thompson, 1683
1	MS 1106	C. Jennings, 1684
1	MS 1152	T. Pointon, 1352
1	MS 1189	K. Cochrane, 1641
1	MS 1189	R. Lemke, 1641
1	MS 0899	Technical Library, 9536 (electronic copy)

v1.40

Sandia National Laboratories